https://doi.org/10.2352/EI.2026.38.8.IQSP-253

An Open-Source Python Implementation of SFRMAT5 for ISO 12233 Spatial Frequency Response Measurement

Robin Jenkin, NVIDIA, USA, Vijayalakshmi Sajjanar, Aditya Khatawkar, Ayaan Dhamnekar, Apeksha Bannigidad, Nishchay Gaonkar, KLE Technological University, India

Abstract

ISO 12233 defines the slanted-edge technique for estimating the spatial frequency response (SFR) of imaging systems. SFRMAT, originally developed by Burns, has served for over two decades as the de facto MATLAB implementation of this standard. This paper presents a fully open-source Python implementation of SFRMAT5 with equivalent functionality and numerical fidelity. The software mirrors the original MATLAB structure to simplify maintenance and coexistence, while supporting CLI, GUI, and headless operation. Validation against the MATLAB version demonstrates practical numerical equivalence ($< 10^{-10}$) on real captures where valid outputs are available. The implementation enables broader adoption in research, education, and production imaging pipelines and is available at `burnsdigitalimaging.com/software/sfrmat`.

Keywords: SFRMAT5, ISO 12233, Spatial Frequency Response, MTF, Python, MATLAB, Open Source

Introduction

Image sharpness is a fundamental attribute in the evaluation of camera and imaging system performance and may be characterized by the Modulation Transfer Function (MTF) [1]. Practical measurement is enabled by the Spatial Frequency Response (SFR) defined in ISO 12233 [2]. The slanted-edge method utilizes a tilted edge followed by projection and super-sampling to obtain an over-sampled Edge Spread Function (ESF). Calculation of the SFR then proceeds via differentiation to obtain the Line Spread Function (LSF) and Fourier analysis to produce the final frequency response. The SFR is distinguished from MTF by virtue that no correction is explicitly made for the spatial frequency content of the target and the capture technique. ISO 12233 is arguably the most commonly used method for evaluating the spatial frequency response of systems today.

A trusted and long-standing implementation of this methodology is SFRMAT, developed in MATLAB [3] by Burns [4], with associated methodological refinements introduced in collaboration with Williams [5]. SFRMAT has closely followed the development of ISO 12233 standard and at the time of writing is on version 5 [6]. It has been widely adopted for camera characterization due to its accuracy, transparency, and reproducibility. The dependence on MATLAB introduces a practical limitation because of the proprietary nature and licensing cost of the platform.

In the last decade, the popularity of Python [7], as an open source and highly extensible programming language, has grown significantly. Python has become a dominant platform for scientific computing, offering a rich ecosystem for numerical analysis, visualization, automation, and integration with machine learning and cloud-based evaluation pipelines. Further, Python enables seamless integration into many automated testing environments and continuous integration frameworks.

Therefore, to extend accessibility, particularly in educational environments, open research initiatives, and cost-sensitive development workflows, this work presents a complete implementation of SFRMAT5 in Python using open-source scientific libraries such as NumPy, SciPy, and Matplotlib. These libraries provide the numerical and visualization capabilities required to preserve the original mathematical logic and computational flow of SFRMAT5 while removing dependency on commercial software. The Python implementation is intentionally structured to mirror the organization of the MATLAB codebase, simplifying verification, maintenance, and long-term coexistence of both versions.

The remainder of this paper documents the motivation, design, and validation of the Python implementation of SFRMAT5. Numerical fidelity is established through direct comparison with the MATLAB reference implementation, including careful handling of platform-specific behaviors that influence polynomial fitting, indexing, and frequency-domain operations. The proposed tool reproduces the reference MATLAB results with high accuracy, exhibiting practical numerical equivalence ($< 10^{-10}$) where valid output exists. By delivering a modular, open-source, and standards-compliant solution, this work lowers the barrier to objective image sharpness evaluation while remaining as faithful as possible to the ISO 12233 methodology.

Background and Related Work

Objective measurement of image sharpness has been an active area of research for several decades, driven by the need for standardized and repeatable evaluation of cameras, scanners, and imaging systems. Early approaches to Modulation Transfer Function (MTF) measurement relied on sinusoidal test patterns [8], bar targets [9], or point sources [1]; however, these methods were often sensitive to noise, alignment errors, and sampling limitations such as aliasing [10]. To address these challenges, Reichenbach et al. proposed the sloping edge technique as a practical methodology to overcome the challenges of aliasing and application to digital systems [11]. By introducing a slight angle between the edge and the pixel grid, the method enables oversampling of the Edge Spread Function (ESF) by projection of the pixel positions in the direction of the edge, allowing accurate recovery of the Line Spread Function (LSF) and its corresponding frequency response, Figure 1.

This methodology formed the original basis for ISO 12233 and has been subsequently enhanced over time to account for various distortions and artifacts that can occur during the imaging

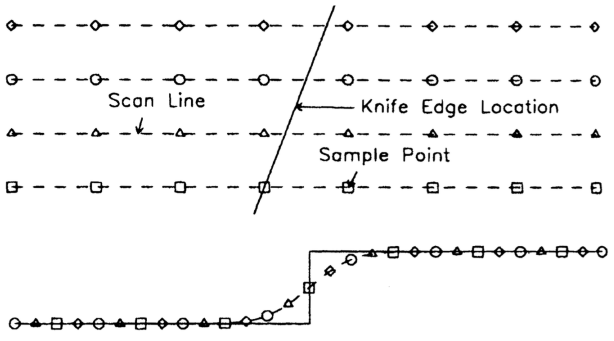


Figure 1. *A visualization of the sloping edge technique of Reichenbach et al. Reproduced from reference [11].*

process.

The accompanying versions of SFRMAT provided a practical and transparent MATLAB-based implementation of the ISO 12233 slanted-edge method. Its algorithmic clarity, numerical reliability, and close adherence to the standard led to widespread adoption in academic research, industrial camera evaluation, and further standards development activities. As a result, SFRMAT has effectively served as a reference implementation against which other tools and measurement pipelines are frequently compared.

A number of commercial and proprietary image quality analysis tools also implement slanted-edge SFR or MTF measurements as part of broader evaluation suites. While these tools are often feature-rich and optimized for production use, their closed-source nature limits transparency, reproducibility, and the ability to inspect or modify underlying algorithms. In addition, licensing costs and platform constraints can restrict accessibility in educational, research, and open development environments.

Despite the growing popularity of Python as a scientific computing platform, prior efforts in this space have largely focused on partial or simplified implementations of edge-based MTF estimation rather than full reproductions of established reference tools. To the best of the authors' knowledge, no fully open-source Python implementation exists that reproduces the complete functionality, numerical behavior, and workflow of SFRMAT5 while providing validated equivalence to the MATLAB reference and adherence to the ISO 12233 methodology.

The work presented in this paper addresses this gap by providing a comprehensive, open-source Python implementation of SFRMAT5 that mirrors the original MATLAB structure and computational pipeline. By preserving algorithmic transparency and establishing numerical equivalence through direct validation, this implementation enables standardized sharpness evaluation without reliance on proprietary software, supporting reproducible research and modern imaging system development. This work is developed using MATLAB 2024b [3] and Python 3.12.7 [7].

Approach and System Architecture

The implementation was approached as a number of logical steps, Figure 2. After analysis of the existing code base and functionality, conversion of each individual SFRMAT5 MATLAB function was completed. Unit testing of those individual components was undertaken before integration into the full Python SFRMAT5 pipeline. Approximately 138 images of edges of various sizes and orientations were acquired and cropped as a validation data set. After processing by both the MATLAB and Python versions of the SFRMAT5 code, comparative graphs of the output of each are created followed by analysis for overall accuracy.

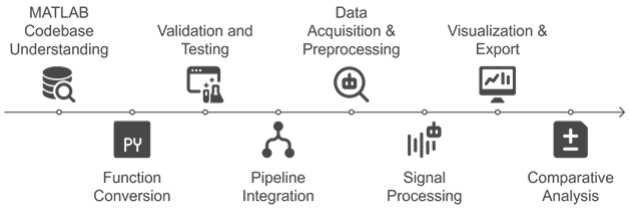


Figure 2. *Implementational approach to conversion of SFRMAT5.*

The Python-based SFRMAT5 implementation is designed as a modular system that closely mirrors the structure and execution flow of the original MATLAB version while enabling flexible integration into modern imaging workflows. As per the original SFRMAT5, the architecture separates image I/O, signal processing, numerical analysis, and visualization into well-defined components, allowing individual modules to be developed, tested, and maintained independently. Individual Python functions may then be updated to mirror any of those made to the original MATLAB in the future.

Figure 3 provides a high-level overview of the main functional elements of SFRMAT5. Core numerical and signal-processing components, including edge localization, ESF projection, derivative filtering, windowing, and frequency-domain correction, are isolated from user interaction and visualization logic. This separation supports consistent numerical behavior across execution modes. By maintaining similar function naming conventions and execution order, numerical discrepancies could be isolated, analyzed, and corrected at the module level without affecting the overall workflow and moving beyond merely replicating functionality.

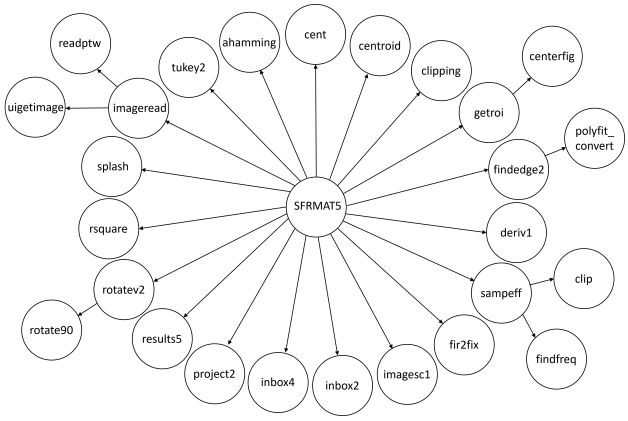


Figure 3. *Overview of the main functional elements of SFRMAT5.*

- **sfrmat5:** Main module coordinating the calculation flow.
- **Mathematical and Signal Processing Modules**
  - **ahamming:** Generates an asymmetric Hamming window centered around a point. It windows the edge

profile before the Fast Fourier Transform (FFT), reducing spectral leakage and improving SFR accuracy in slanted-edge analysis.
  - **cent:** Shifts the elements of a 1D signal such that a specific index (typically the edge centroid) is centered in the array. This ensures proper alignment of the edge spread function (ESF) for downstream LSF and FFT.
  - **centroid:** Computes the center of mass of a 1D array, crucial for determining the precise location of the edge transition. This helps in centering the ESF before applying window functions or derivatives.
  - **clip:** Constrains an array's values within a defined min-max range. Used to avoid extreme values during normalization and visualization, ensuring the numerical stability of computations.
  - **clipping:** Checks for saturation artifacts in the image (pixel values at 0 or max), which could distort SFR measurement. It flags such cases so the user can inspect and validate input data quality.
  - **deriv1:** Calculates the first derivative of the ESF to obtain the LSF. This is a vital step before performing the FFT for computing the modulation transfer function.
  - **fir2fix:** Applies a frequency correction factor to the SFR curve to compensate for the smoothing loss introduced by derivative filters like deriv1.
  - **findfreq:** Detects the spatial frequency at which the normalized SFR falls below a specified threshold (commonly 0.1 or 0.5), helping determine system resolution and image sharpness metrics.
  - **rsquare:** Computes R2 and RMSE to evaluate polynomial fitting accuracy of the edge profile. Ensures the edge fitting models (used in ESF projection) are valid and reliable.
  - **sampeff:** Calculates the sampling efficiency of the system by comparing actual SFR to the ideal sine function. Helps evaluate how close the imaging system is to the Nyquist limit.
- **Image Processing and ROI Modules**
  - **getroi:** Enables manual or programmatic selection of a Region of Interest (ROI) from the input image. This is the initial and critical step for isolating the edge pattern used in SFR analysis.
  - **findedge2:** Uses polynomial regression to trace the slanted edge within the ROI, returning the angle and coordinates. This edge information is crucial for rotation and projection steps.
  - **rotate90 / rotatev2:** Rotates the ROI or image such that the slanted edge becomes vertically aligned. This simplifies ESF extraction and ensures consistent orientation during batch processing.
  - **project2:** Projects 2D slanted-edge data into a 1D edge spread function (ESF) using orthogonal projection. It aligns the edge data along a normalized axis before derivation and FFT.
- **I/O and Visualization Modules**
  - **imageread:** Handles multi-format image reading, including TIFF, PNG, JPEG, and other formats. It converts them into consistent NumPy arrays for unified processing.
  - **readptw:** Specialized reader for FLIR thermal images stored in . ptw or . ptm format. Converts them into 3D arrays, enabling SFR computation even on infrared image datasets.
  - **uigetimage:** GUI-based file selector for choosing input images. Improves ease-of-use, especially in batch-mode or during GUI-driven SFR sessions.
  - **centerfig:** Repositions output figures to the center of the screen for better visibility. It enhances user interaction during visualization of ESF, LSF, and SFR plots.
  - **imagesc1:** Displays grayscale or RGB images with autoscaling for contrast. Used to preview the input image, ROI, edge detection result, and SFR output visually.
  - **splash:** Displays an initial splash screen with version information and credits when the pipeline starts. Mainly used for UI friendliness and identification of the tool version.
  - **results5:** Saves the final computed metrics - including SFR curves, sampling efficiency, and edge data - to an Excel-compatible file for documentation and comparison.
- **Utility and Parameter Modules**
  - **inbox2, inbox4:** Popup dialogs used to input key parameters such as sampling interval, polynomial order, and luminance weights. These parameters greatly influence the fitting and SFR results.
  - **polyfit_convert:** Converts polynomial coefficients from normalized coordinates to image pixel units. This makes the polynomial fit results interpretable in real-world terms.
- **Testing Module**
  - **test_sfrmat5:** A master test script to validate the entire pipeline. Simulates a full workflow from image selection to SFR plotting, ensuring all modules function as intended after migration.

The system supports multiple image formats commonly used in camera evaluation workflows. Input images are converted into consistent NumPy array representations to standardize downstream processing. Region-of-interest (ROI) selection is performed either interactively or programmatically, enabling both exploratory analysis and automated batch processing. Preprocessing steps include clipping, saturation detection, orientation correction, and centroid estimation to ensure accurate edge alignment prior to SFR computation.

At the core of the architecture is the slanted-edge processing pipeline responsible for transforming image data into spatial frequency response measurements. The core executes independently of the user interface and implements the numerical operations defined by the ISO 12233 slanted-edge methodology. By isolating the computational core, the implementation supports deterministic execution and reproducible results across platforms and execution environments.

To support diverse usage scenarios, the system provides three execution modes as per the original software:

- **Command-line interface (CLI):** Enables scripted execution and batch processing for large datasets or automated testing pipelines.
- **Graphical user interface (GUI):** Provides interactive tools for image selection, ROI adjustment, parameter tuning, and visualization of intermediate and final results.
- **Headless mode:** Allows integration into larger imaging systems, continuous integration frameworks, or cloud-based evaluation environments without graphical dependencies.

All execution modes invoke the same underlying processing core, ensuring identical algorithmic behavior regardless of how the tool is operated.

Visualization components generate plots of the ESF, LSF, and SFR curves for individual color channels and luminance, closely matching the appearance and interpretation of the MATLAB reference output. Final results, including SFR data and derived metrics such as sampling efficiency, can be exported in tabular formats suitable for documentation, comparison, and further analysis.

The architectural emphasis on modularity and structural parity with the MATLAB implementation simplifies long-term maintenance and extension. New features—such as additional image formats, automated ROI detection, or alternative visualization backends—can be incorporated without altering the validated computational core. This design ensures that the Python implementation remains faithful to the reference behavior while being adaptable to evolving imaging system requirements.

Methodology

As previously mentioned, the Python implementation of SFRMAT5 was developed through a structured, modular translation of the original MATLAB codebase, with the primary objective of preserving algorithmic behavior, numerical accuracy, and adherence to the ISO 12233 slanted-edge methodology. The implementation follows the same conceptual processing pipeline as the reference version while leveraging open-source Python libraries to ensure portability and reproducibility.

Development began with a detailed analysis of the MATLAB SFRMAT5 codebase to understand its data flow, function dependencies, and numerical assumptions. Particular attention was paid to edge detection, centroid estimation, polynomial fitting, windowing, derivative filtering, and frequency-domain normalization, as these stages are critical to SFR accuracy. MATLAB-specific behaviors that could influence numerical equivalence were identified early to guide the Python translation.

Each MATLAB function was translated into an equivalent Python module using NumPy [12] and SciPy [13], preserving the original mathematical formulations and execution order.

The computational pipeline follows the ISO 12233 slanted-edge methodology. Input images containing a slanted edge target are first imported and preprocessed. A region of interest (ROI) is selected and oriented such that the edge is approximately vertical. The edge location is estimated using polynomial regression, and the ESF is constructed by projecting pixel values orthogonal to the detected edge.

The ESF is smoothed and differentiated to obtain the LSF, which is subsequently windowed to reduce spectral leakage. A discrete Fourier transform of the LSF yields the spatial frequency response, which is normalized and corrected for derivative filtering effects to produce the final SFR curves for individual color channels and luminance.

All translated modules are integrated into a unified processing workflow that supports command-line, graphical, and headless execution. This flexibility enables interactive analysis, batch processing, and automated testing within larger imaging pipelines. Input parameters such as ROI selection, polynomial order, and channel weighting are configurable, matching the behavior of the MATLAB reference implementation.

Several MATLAB-specific behaviors required explicit handling in Python to support numerical alignment. These included differences in array indexing conventions, floating-point rounding behavior in FFT-based operations, and window generation for asymmetric Hamming functions. In the case of polynomial fitting, MATLAB's internal normalization of the independent variable was replicated explicitly in Python, including equivalent mean and standard deviation scaling, followed by conversion of fitted coefficients back to the original coordinate system. These adjustments were necessary to enable reproducible alignment between implementations, which was subsequently confirmed through validation.

Validation and Numerical Fidelity

A central objective of this work was to ensure that the Python implementation of SFRMAT5 reproduces the numerical behavior of the MATLAB reference implementation with high fidelity. Validation was therefore performed at both module and system levels, using identical input images, parameter settings, and processing configurations across both platforms.

Validation focused on direct numerical comparison of intermediate and final outputs rather than visual agreement alone. For a given input image and region of interest, the Python and MATLAB implementations were executed using the same polynomial order, windowing parameters, and channel weighting. Outputs were compared at key stages of the pipeline, with particular emphasis on the final spatial frequency response (SFR) curves. Unit testing of each individual function was completed.

Particular attention was given to polynomial fitting behavior, as small deviations in edge modeling can propagate through the ESF projection and frequency-domain analysis. To quantify agreement between implementations, several numerical consistency metrics were evaluated:

- **Pointwise absolute SFR difference:** The absolute difference between MATLAB and Python SFR values was computed at each spatial frequency sample.
- **Maximum absolute deviation:** The maximum observed difference across the frequency axis was recorded for each color channel and luminance.
- **Cross-channel consistency:** Differences were evaluated independently for the red, green, blue, and luminance SFR outputs.

These metrics enabled precise identification of discrepancies and guided targeted corrections during development.

Polynomial Fitting Consistency and Indexing

Accurate edge localization is critical to the slanted-edge method, as small deviations in polynomial fitting can propagate into the ESF projection and subsequent frequency-domain analysis. MATLAB's `polyfit` internally normalizes the independent variable using its mean and standard deviation, where the standard deviation is computed with normalization by N rather than $N-1$ [3].

To replicate this reference behavior, the Python implementation explicitly computes the normalization parameters using `numpy.std` with `ddof=0`, ensuring equivalence with MATLAB's default behavior. Polynomial coefficients are then converted back to the original coordinate system using a dedicated coefficient conversion routine. This explicit handling avoids hidden normalization differences and ensures consistent edge localization between platforms.

Additional numerical differences were traced to indexing conventions. MATLAB arrays are 1-indexed, while Python arrays are 0-indexed, which can affect frequency index interpretation in threshold-based measurements. In particular, the frequency detection routine required an explicit index offset correction to align Python results with MATLAB outputs. Applying this correction eliminated off-by-one discrepancies in reported frequency metrics.

Results and Discussion

The Python implementation of SFRMAT5 was evaluated through direct numerical comparison with the original MATLAB reference using identical input images, regions of interest, and processing parameters. Agreement was assessed across the full spatial frequency range for the red, green, blue, and luminance channels.

Across all evaluated cases, the Python implementation demonstrated near-identical agreement with the MATLAB results. The practical difference is less than 10^{-10} between valid MATLAB outputs and the Python version. Images that produced MATLAB outputs of NaN were not compared. These results confirm that the Python implementation faithfully reproduces the numerical behavior of the reference system.

Figures 4 and 5 show an overlay of the SFR curves produced by the MATLAB and Python implementations. The close correspondence across the spatial frequency axis indicates consistent edge localization, ESF projection, frequency-domain processing, and normalization between the two platforms.

Overall, the results demonstrate that the Python-based SFRMAT5 implementation achieves numerical equivalence with the MATLAB reference while providing the advantages of an open-source, platform-independent solution suitable for research, education, and production imaging workflows.

Limitations and Future Work

While the Python implementation of SFRMAT5 achieves high numerical fidelity with the MATLAB reference, several limitations remain. Support for `.rgv` and `.stv` is not available as an open library to the authors' knowledge. Additionally, whilst `.ptm`, and `.ptw` image formats are available, they have not been extensively tested due to a lack of representative sample data for validation. There is no reason to believe that the function will fail for these formats and, as additional datasets become available, the processing core and image loading can be validated.

Although explicit normalization and indexing corrections were implemented to replicate MATLAB's internal `polyfit` behavior, polynomial fitting remains a numerically sensitive stage of the slanted-edge pipeline. While the remaining differences are negligible in practice, further investigation into edge fitting robustness under extreme noise or low-contrast conditions may provide additional insight.

Future development efforts will focus on enhancing usability and automation. Planned improvements include automatic region-of-interest detection, expanded batch processing support, and refinements to the graphical user interface. Integration with cloud-based analysis environments and continuous integration pipelines is also envisioned, enabling scalable and automated image quality evaluation workflows.

These enhancements aim to extend the applicability of the tool while preserving strict adherence to the ISO 12233 methodology and the validated numerical behavior established in this work.

Conclusion

This work presents a complete open-source Python implementation of SFRMAT5 that faithfully reproduces the numerical behavior and functional structure of the original MATLAB reference. Through explicit handling of normalization, indexing, and polynomial fitting behavior, the implementation achieves numer-

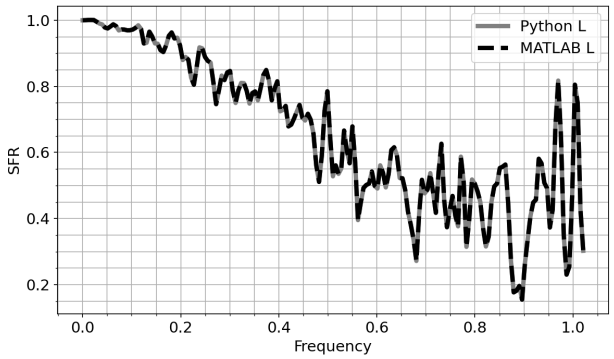


Figure 4. *Example results from MATLAB and Python SFRMAT5 for a greyscale image.*

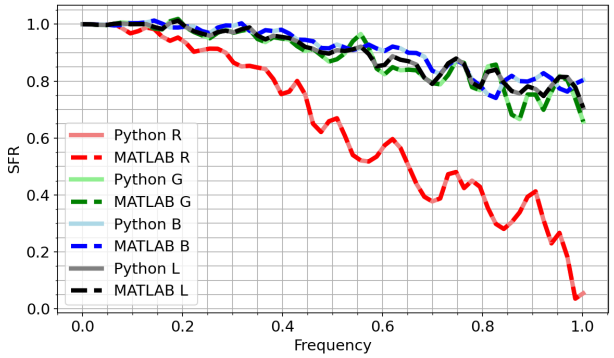


Figure 5. *Example results from MATLAB and Python SFRMAT5 for an RGB image.*

ical precision agreement across spatial frequencies while maintaining transparency and reproducibility.

By removing dependence on proprietary software, the Python-based SFRMAT5 lowers the barrier to standardized image sharpness evaluation and enables broader adoption in research, education, and production imaging workflows. Feedback from the original author of SFRMAT5 further confirms the value of this contribution to the image quality assessment community.

The software is intended for public release in conjunction with this work, providing an accessible and extensible reference implementation aligned with the ISO 12233 standard.

Acknowledgments

The authors wish to thank Peter Burns, Burns Digital Imaging, and the original author of SFRMAT, who has been very supportive of this work and has shared valuable knowledge and insight freely. We would also like to thank Balaji Holur, NVIDIA, who established the KLE mentorship program under which this work was undertaken.

References

[1] G. D. Boreman, *Modulation Transfer Function in Optical and Electro-Optical Systems*, 2nd ed., SPIE Press, Bellingham, WA (2021).

[2] ISO 12233:2024, Digital cameras — Resolution and spatial frequency response (SFR), International Organization for Standardization, Geneva, Switzerland (2024).

[3] The MathWorks Inc., *MATLAB version 24.2 (R2024b)*, Natick, Massachusetts (2024).

[4] P. D. Burns, "Slanted-edge MTF for digital camera and scanner analysis," In: *Proc. IS&T PICS Conference*, pp. 135–138 (2000).

[5] P. D. Burns and D. Williams, "Refined slanted-edge measurement for practical camera and scanner testing," In: *Proc. IS&T PICS Conference*, pp. 191–195 (2002).

[6] P. D. Burns, "ISO 12233 sfrmat5," Burns Digital Imaging (2024). `burnsdigitalimaging.com/software/sfrmat/iso12233-sfrmat5`

[7] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA (2009).

[8] J. W. Coltman, "The specification of imaging properties by response to a sine wave input," *J. Opt. Soc. Am.*, 44(6), pp. 468–471 (1954).

[9] D. N. Sitter, Jr., J. S. Goddard, and R. K. Ferrell, "Method for the measurement of the modulation transfer function of sampled imaging systems from bar-target patterns," *Appl. Opt.*, 34(4), pp. 746–751 (1995).

[10] S. K. Park, R. Schowengerdt, and M. Kaczynski, "Modulation-transfer-function analysis for sampled image systems," *Appl. Opt.*, 23(15), pp. 2572–2582 (1984).

[11] S. E. Reichenbach, S. K. Park, and R. Narayanswamy, "Characterizing digital image acquisition devices," *Opt. Eng.*, 30(2), pp. 170–177 (1991).

[12] C. R. Harris, K. J. Millman, S. J. van der Walt, et al., "Array programming with NumPy," *Nature*, 585, pp. 357–362 (2020).

[13] P. Virtanen, R. Gommers, T. E. Oliphant, et al., "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat. Methods*, 17, pp. 261–272 (2020).

Author Biography

Robin Jenkin received a BSc(Hons) in Photographic and Electronic Imaging Science (1995) and his PhD (2001) in the field of image science from the University of Westminster. He also holds an M.Res in Computer Vision and Image Processing from University College London (1996). Robin is a Fellow of The Royal Photographic Society, UK, and Executive Vice President of the Society for Imaging Science and Technology. Robin is secretary of the IEEE P2020 Image Quality for Autonomous Vehicles working group. At NVIDIA, Robin is a Distinguished Engineer and models image quality for autonomous vehicles and other applications. He is a Visiting Professor at the University of Westminster within the Computer Vision and Imaging Technology Research Group and co-author of the 10th edition of The Manual of Photography, Focal Press.

Ayaan Dhamnekar is a final-year undergraduate student pursuing a B.E. in Computer Science and Engineering at KLE Technological University, Hubballi, India (expected 2026). He served as a Project Intern under NVIDIA's AI Technology Center (NVAITC) program, contributing to the open-source Python implementation of SFRMAT5 described in this paper. He has hands-on experience in full-stack web development (frontend, backend services, and database-backed deployments) and is currently a software developer intern in an FX Company. His technical interests span image processing, scalable web systems, applied machine learning, and the creation of reliable software systems and developer tools.

Aditya Khatawkar is a final-year undergraduate student in Computer Science and Engineering at KLE Technological University, Hubballi, India. He is currently a Software Development Intern at NANTECH Solutions, where he develops mobile applications using Flutter. His technical interests include problem-solving and competitive programming, with a strong focus on algorithmic thinking and data structures. Additionally, he is passionate about backend web development, Python programming, and artificial intelligence.

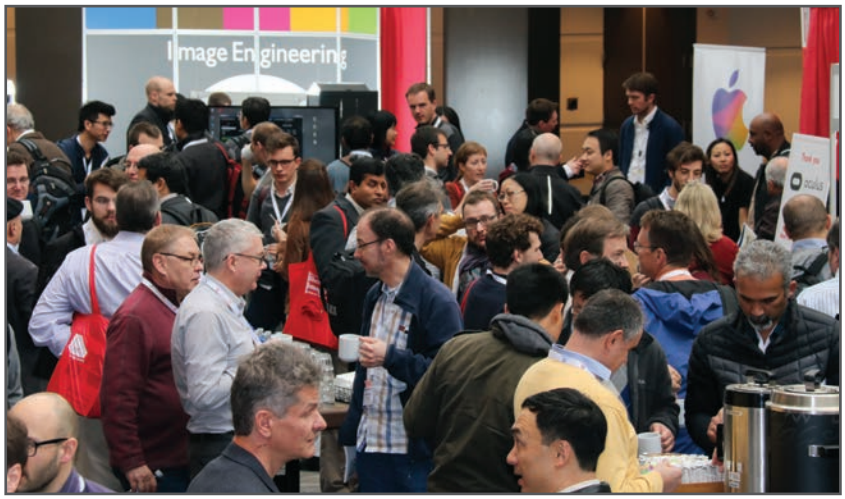
Apeksha Bannigidad is a final-year undergraduate student in Computer Science and Engineering at KLE Technological University, Hubballi, India. She is currently working as a Software Development Intern at Dvara Group, where she focuses on developing real-time analytical dashboards using Looker Studio and working with data warehousing technologies such as BigQuery. Her research interests include full-stack web development, machine learning, and artificial intelligence. She has a strong interest in problem-solving and continuously enhances her analytical thinking skills through the study of data structures and algorithms.

Nishchay M. Gaonkar is a final-year undergraduate student in Computer Science and Engineering at KLE Technological University, Hubballi, India. He is currently working as a Software Developer Intern at NPCIL, Kaiga, where he is involved in developing analytical and predictive software solutions for nuclear reactor operations. His current work focuses on modeling and predicting xenon override conditions and poison-out time in nuclear reactors using data-driven approaches. His technical interests include Python programming, machine learning, and data analysis, along with full-stack web development.

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org